Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Albin Jordan

Pridobivanje podatkov iz računov s pomočjo NFC

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Rok Rupnik

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:
Prejemanje računov s pomočjo protokola NFC (Acquiring data from bills through NFC)

Tematika naloge:

V okviru diplomskega dela izdelajte sistem za prejemanje računov na mobilno napravo Android s pomočjo protokola NFC. Poleg mobilne aplikacije izdelajte tudi spletni vmesnik, ki uporabnikom dovoljuje pregled in analizo prejetih računov. Za proces načrtovanja uporabite agilno metodologijo SCRUM. V zaključnem delu predstavite prednosti tehnologije NFC in opišite možne nadgradnje razvitega sistema.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Albin Jordan z vpisno številko 63110172 sem avtor diplomskega dela z naslovom:

Pridobivanje podatkov iz računov s pomočjo NFC (angl. *Acquiring data from bills through NFC*)

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Roka Rupnika,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) in ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu prek univerzitetnega spletnega arhiva.

V Ljubljani, dne 27. februarja 2016 Podpis avtorja:

Zahvaljujem se mentorju doc. dr. Roku Rupniku za pomoč in svetovanje pri izdelavi diplomskega dela.

Zahvalil bi se tudi sestri Karmen J., ki mi je z veseljem priskočila na pomoč pri lektoriranju in strukturiranju diplomskega dela.

Posebno pa bi se zahvalil dekletu Karmen P. za vso izkazano potrpljenje in spodbudo med nastajanjem dela.

Svoji dragi Karmi.

Kazalo

Slike

Seznam uporabljenih kratic

kratica	angleško	slovensko
NFC	Near Field Communication	komunikacija v bližnjem polju
IR	Infrared light	infrardeča svetloba
JPA	Java Persistence API	ogrodje za shranjevanje objektov
MVC	model, view, controller	model, pogled, kontroler
REST	Representational State Transfer	arhitektura za izmenjavo podatkov
JSP	Java Server Pages	ogrodje za javanske strani
HTTP	HyperText Transfer Protocol	metoda za prenos informacij
POJO	plain old java object	osnovni javanski objekt
API	Application Programming Interface	zunanji vmesnik
SOAP	Simple Object Access Protocol	arhitektura za izmenjavo podatkov
JMX	Java Management Extensions	javanski vtičnik za nadzor
URI	Uniform Resource Identifier	enolični identifikator zahtevka
XML	Extensible Markup Language	format za opis podatkov
JSON	JavaScript Object Notation	format za opis podatkov
CRUD	create, read, update, delete	ustvari, preberi, posodobi, izbriši
HTML	Hyper Text Markup Language	definicija prikaza vsebine
SVG	vector-based graphics	značke za vektorsko risanje
AJAX	Asynchronous JavaScript and XML	asinhrono pridobivanje podatkov
SCRUM	agile software development framework	proces agilnega programiranja
JAXB	Java Architecture for XML Binding	mapiranje XML v POJO in obratno

Povzetek

Naslov: Pridobivanje podatkov iz računov s pomočjo NFC

V diplomskem delu sta predstavljeni implementacija sistema za pridobivanje računov na mobilno napravo s pomočjo protokola NFC in sinhronizacija računov s spletno stranjo.

Prvo poglavje opisuje razvoj in uporabo protokola NFC, obenem pa predstavlja njegove varnostne luknje in možnosti preprečitve napadov.

Sledi drugo poglavje, v katerem so opisana uporabljena orodja in tehnologije, ki so bile potrebne za razvoj sistema. Večina opisov tehnologij vključuje tudi preprost primer, ki omogoča lažje razumevanje vsebine.

Glavni del diplomskega dela je zajet v tretjem poglavju, ki opisuje proces izgradnje sistema. V prvem delu sta opisana strežniški del aplikacije in komunikacija s podatkovno bazo. Sledi predstavitev izgradnje mobilne aplikacije in podoba, dodani pa so še odseki poslovne logike komunikacije NFC. V naslednjem delu se podrobneje seznanimo z izdelavo namizne aplikacije, ki simulira blagajno in pošilja naključno generirane račune. Poglavje zaključuje opis izgradnje spletnega dela, ki zajema sinhronizacijo podatkov in njihov prikaz v brskalniku.

V zaključku so opisane ugotovitve, pridobljene med razvojem aplikacije, in možnosti nadgradnje obstoječega sistema.

Ključne besede: račun, NFC, spletna storitev, Android

Abstract

Title: Acquiring data from bills through NFC

The main idea of the following thesis is to present the development of the system used for receiving paper receipts on a mobile machine with the usage of the NFC protocol.

In the first chapter we familiarize ourselves with the use and evolution of the NFC protocol, we present main security holes and methods to prevent them from happening.

The second chapter is reserved for technologies and development environments used in our system. Most technologies are also backed by simple examples which provide easier understanding.

The main part of the thesis is chapter three which describes development parts and the agile process used for achieving faster results. First part of chapter three describes the server side of the application with persistence layer. We continue with the construction of the mobile application. We present its design and add code samples of the NFC adapter integration. Third subsection presents the desktop application which generates random receipts and forwards them through the NFC protocol. The chapter is concluded with web application development.

Finnally we wrap up the thesis with our thoughts on the system and we provide some ideas for possible future extensions.

Keywords: receipt, NFC, webservice, Android

Poglavje 1

Uvod

Živimo v svetu, v katerem programsko opremo srečujemo na vsakem koraku, kar nam omogočata uporaba interneta in množični porast uporabe elektronskih naprav za zasebno rabo. Vseeno pa menimo, da je na nekaterih področjih izkoriščanje možnosti, ki jih ponujajo elektronske naprave za zasebno rabo, še vedno precej slabo. Pri opravljanju vsakodnevnih nakupov tako še zmeraj prejemamo račune, natisnjene na papir.

Papirnati računi so za mnoge uporabnike veliko breme, še posebej sedaj, ko se morajo zaradi možnosti dokazovanja davčni upravi vedno hraniti na hitro dostopnem mestu. Poleg tega so nujno zlo pri izdajanju in uveljavljanju garancijskih listov. Dodatna slabost papirnatih računov pa je, da kupcem ne omogočajo hitrega pregleda nakupov. Za podrobnejše analize osebnih financ je treba posegati po tuji programski opremi, v katero mora uporabnik podatke z računa vnašati ročno, kar pa je dostikrat prezahtevno.

Tudi prodajalcem povzročajo papirnati računi precej težav. Mesečno mora večja trgovska veriga za nakup papirja in črnila odšteti kar 10 000 €, pri tem pa se moramo zavedati, da se letno zaradi tega poseka na stotine dreves.

Zakaj ne bi vsega tega nadomestili s preprosto aplikacijo, ki bi omogočala prejemanje računov kar na pametno napravo? Za prenos podatkov bi uporabljali protokol NFC, ki je že prisoten v večini trgovin (brezstično plačevanje).

Mobilno napravo bi samo približali terminalu in vsebina računa bi se prenesla. To bi nam omogočalo hitro iskanje računov, njihovo filtriranje in večjo preglednost. Primerjali bi lahko cene izdelkov v posameznih trgovinah, vodili osebno evidenco in imeli garancijske liste vedno pri roki. Poleg tega bi se mobilna aplikacija sinhronizirala s spletno aplikacijo, ki bi ponujala dodatne funkcije. Uporabnik bi lahko določal maksimalno mesečno porabo v določeni trgovini, se poigraval s statistiko nakupov in si nastavljal opomnike za želeni mesečni nakup.

V aplikacijo smo želeli vključiti vse omenjene funkcije. Sprva smo se podrobneje seznanili s protokolom NFC in javanskimi knjižnicami, potrebnimi za razvoj. V drugem poglavju smo na kratko opisali vsa uporabljena orodja, tehnologije, strežnike in platforme, ki omogočajo izgradnjo aplikacije. V osrednjem delu pa smo predstavili načrtovanje s pomočjo agilne metodologije SCRUM, arhitekturo baze in podrobneje opisali sam razvoj aplikacije.

Na koncu smo predstavili pridobljene ugotovitve in zaključili z opisom možnih nadgradenj sistema.

Poglavje 2

NFC – Near Field Communication

NFC je skupek komunikacijskih protokolov, ki omogočajo prenos podatkov med dvema elektronskima napravama [1]. Navadno je ena izmed naprav pametni telefon, druga pa je preprost NFC-čip ali NFC-značka, seveda pa so mogoče tudi druge kombinacije povezovanja, kar bomo spoznali v nadaljevanju. Maksimalna razdalja med napravama za uspešno komunikacijo je 4 centimetre, kar je precej manj kot pri sorodnih protokolih (Bluetooth, IR, WI-FI itd.). S tem je zagotovljena bistveno večja varnost, kar omogoča uporabo protokola pri varnostno občutljivejših podatkih (plačevanje, zdravstvo itd.). Trenutno je protokol NFC najpogosteje uporabljen ravno pri plačevanju, vse pogosteje pa njegovo učinkovitost izrabljajo transportne in oglaševalske industrije.

Naprava, ki podpira protokol NFC, omogoča eno izmed naslednjih konfiguracij za komunikacijo:

- emulacija kartice (omogoča, da se naprava predstavi kot kartica, kar je med drugim uporabno pri plačevanju oziroma nakupovanju vstopnic),
- branje/pisanje (omogoča napravi branje ali pisanje na želen NFC-medij),
- P2P – vsak z vsakim (omogoča neprekinjeno komunikacijo med napra-

vama).

2.1 Vrste možnih napadov

Zaradi uporabe brezstične tehnologije pri varnostno občutljivih podatkih se pojavlja vprašanje varnosti. NFC-tehnologija zaradi omejevanja razdalje za uspešno komunikacijo mnoge napade uspešno odbija, vseeno pa obstajajo določene nevarnosti. Nekaj možnih napadov je podrobneje opisanih v nadaljevanju [2].

2.1.1 Prisluškovanje (angl. eavesdropping)

V primeru prisluškovanja napadalec uporabi posebno anteno, s katero posname komunikacijo med napravama. Kljub temu da sta napravi zelo blizu, je napad mogoč. Navadno napadalci s tem pridobijo podatke, ki jih prenesejo na novo napravo, in z njimi simulirajo identiteto ukradenega uporabnika. Preprečevanje tovrstnih napadov poteka s pomočjo enkripcije, ki poskrbi, da so ukradeni podatki za napadalca neuporabni.

2.1.2 Modifikacija podatkov (angl. data modification)

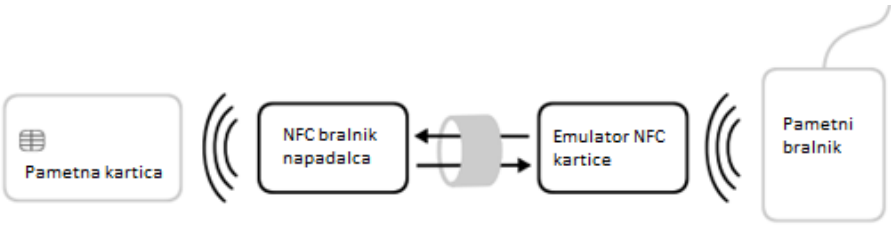
Napad z modifikacijo podatkov je izveden tako, da napadalec prestreže in modificira podatke (binarno kodo) s pomočjo radiofrekvenčne naprave. Običajno se pri tem podatki poškodujejo, zato za uporabnika ni večjih izgub, obstajajo pa primeri, ko napadalec z ustreznim modificiranjem podatkov uspešno vzpostavi povezavo s sprejemnikovo napravo. Da se izognemo tovrstnim napadom, se lahko uporabi posebna programska oprema, ki s preverjanjem radiofrekvenčnega polja zazna napad in preneha s komunikacijo.

2.1.3 Rele napad (angl. relay attack)

Pri izvedbi tako imenovanega rele napada ustvari napadalec virtualno povezavo, se predstavi z ukradeno identiteto in zlorabi žrtvine podatke. Na-

pad poteka tako, da napadalec NFC-bralnik, povezan z zunanjo napravo (računalnik), približa žrtvinemu mediju (telefon/kartica) in po drugem kanalu, ki je v večini primerov internet, posreduje podatke NFC-zapisovalniku na drugi strani kanala. Tako lahko na drugi strani izvede plačilo, pri katerem se predstavlja z žrtvino identiteto.

Najučinkovitejša metoda za preprečevanje tovrstnih napadov je shranjevanje NFC-naprav v posebno zaščitno embalažo (Faradayeva kletka), ki razločuje električno polje in s tem onemogoča vzpostavljanje komunikacije zunanjemu mediju.



Slika 2.1: Rele napad

2.1.4 Lažno predstavljanje (angl. spoofing)

Za izvedbo napada z lažnim predstavljanjem napadalec namesti NFC-čip ali NFC-značko v bližino prave NFC-naprave. Ko uporabnik približa svojo mobilno napravo, ta pogosto zazna napadalčevo. Večina mobilnih naprav v primeru prejetega ukaza tega takoj izvrši in s tem sproži želeno akcijo. To pomeni, da lahko napadalec v značko zapiše URL-naslov, ki na žrtvin telefon namesti zlonamerno kodo. Eden od načinov za preprečevanje tovrstnih napadov je vklop potrjevanja zahtevkov, s katerim si uporabnik zagotovi lastno potrjevanje sproženih operacij.

Poglavje 3

Uporabljena orodja in tehnologije

Za izdelavo mobilne aplikacije je uporabljena platforma Android. Platformo po svetu uporablja kar 70 % vseh mobilnih uporabnikov, gre pa tudi za najpogosteje uporabljeno platformo v Sloveniji. Zaledje informacijskega sistema se nahaja na strežniku Tomcat in je v celoti spisano v programskem jeziku Java. Shranjevanje podatkov je podprto s trenutno najpopularnejšo bazo, in sicer NOSQL-bazo MongoDB. Spletni del aplikacije je prav tako spisan v programskem jeziku Java (EE), MVC-pristop pa smo zagotovili z uporabo tehnologij Spring, Velocity, HTML, JQuery in LESS. Tako za komunikacijo med mobilno aplikacijo in strežnikom kot za komunikacijo med spletno aplikacijo in strežnikom smo uporabili arhitekturni stil REST.

3.1 Android

Začetki operacijskega sistema Android segajo v leto 2005, ko ga je razvilo podjetje Android Inc. Njegova glavna naloga je bila zagotoviti lažje in hitrejše upravljanje mobilnih naprav. Kmalu je velik potencial hitro rastočega podjetja zaslutil računalniški gigant Google in nedolgo po prevzemu ponudil verzijo 1.0, ki je v primerjavi z današnjo situacijo uporabniku ponujala

zelo skop nabor funkcij (budilka, internetni brskalnik in upravljanje kamere). Hiter razvoj tehnologije je tudi operacijskemu sistemu omogočil številne izboljšave. Tako se je do današnjega dne izmenjalo več kot 10 verzij. Najnovejša verzija, ki jo trenutno najdemo na trgu, je Android 6.0 Marshmallow, izdan konec oktobra 2015 [3].

Sistem Android lahko podrobneje definiramo glede na 5 ključnih elementov:

- aplikacije,
- aplikacijsko ogrodje,
- knjižnice,
- prevajalnik,
- Linux Kernel (jedro).

3.2 Aplikacijski strežnik Tomcat

Aplikacijski strežnik Tomcat je razvilo podjetja Apache Software Foundation in ga kot odprtokodno rešitev ponudilo razvijalcem programske opreme. Strežnik skrbi za izvrševanje javanskih servletov (angl. Java Servlet) in izris spletnih strani, ki vključujejo strežniške strani (angl. Java Server Pages). Njegove glavne komponente so Catalina (servletski kontejner), Coyote (HTTP-konektor) in Jasper (izris JSP-jev).

V našem sistemu je uporabljen za sprejemanja in pošiljanje HTTP-zahtevkov.

3.3 Ogrodje Spring

Ogrodje Spring ponuja napredno infrastrukturo z naborom knjižnic, ki močno poenostavljajo razvoj aplikacij, spisanih v programskem jeziku Java. Njegova glavna prednost je zmožnost uporabe preprostih javanskih objektov (POJO)

za izvedbo kompleksnih operacij JavaEE. Naj naštejemo nekaj API-jev, ki so del ogrodja Spring:

- Spring Boot (hitra začetna postavitev spletne aplikacije),
- podatkovni del oziroma baza (povezava in manipulacija podatkov),
- varnost (preprosta avtentikacija in avtorizacija),
- komunikacija med strežnikom in odjemalcem (REST, SOAP),
- sporočilni sistem (JMX),
- pristop MVC (model, pogled, krmilnik).

3.4 MongoDB

Dokumentno usmerjena nerelacijska (NoSQL) baza predstavlja mehanizem za shranjevanje in pridobivanje podatkov [4]. MongoDB zamenjuje tradicionalno tabelarično strukturo relacijske baze z dinamičnimi shemami, ki močno spominjajo na JSON (BSON) [5]. To pomeni, da predhodno načrtovanje tabel in njihova vezava nista več potrebna, saj se lahko dokument, ki je shranjen v kolekcijo, razlikuje od drugega elementa, prav tako vstavljenega v isto kolekcijo.

Primer prikaza dokumenta vrnjenega iz poizvedbe:

```
{
    _id: 1111,
    buyer: { name: "Janez Kranjski", email : "janez@kranjski.com" },
    title: "Mercator",
    date: { $date: "2010-07-12 13:23UTC" },
    location: [ -121.2322, 42.1223222 ],
    articles: [
       { article: "bread",
         price: "10",
```

```
       quantity : 2,
      { article: "water",
        price: "1",
        quantity: 22,
        discount: "10%"
  }
   ],
   tags: [ "Politics", "Virginia" ]
}
```

3.5 REST

V računalništvu se s kratico REST (angl. Representational State Transfer) srečamo pri definiciji arhitekturnega stila za spletno komunikacijo med strežnikom in odjemalcem. Namesto REST pogosto uporabljamo tudi poimenovanje spletna storitev. Prednost arhitekturnega stila REST pred arhitekturnim stilom SOAP (angl. Simple Object Access Protocol) [6] je predvsem v preprostosti in manjši porabi pasovne širine. Glavni principi tehnologije REST so naslednji [7]:

- viri (dostopni s preprosto direktorijsko strukturo URI),
- predstavitev podatkov s pomočjo XML- ali JSON-strukture,
- sporočilnost z uporabo HTTP-metod (GET, POST, PUT in DELETE).

3.5.1 HTTP-metode

Uporaba metod HTTP je zelo priročna pri mapiranju operacij CRUD (create, retrieve, update, delete).

GET

Metoda GET je značilna za pridobivanje informacij. Definira jo idempotentnost, kar pomeni, da se pri njeni uporabi stanje v bazi podatkov ne spreminja. V ozadju se sicer lahko zgodi, da pride do kakšni nezaželenih ukazov/akcij, vendar ti za uporabnika ne predstavljajo sprememb. Zahtevek je lahko delen ali pogojen.

Primer pridobivanja računa z enoličnim identifikatorjem 1:

```
GET /receipt/1
```

POST

Zahtevki, ki vsebujejo glavo POST, so navadno namenjeni manipulaciji podatkovnih entitet. Najpogosteje se uporabljajo za njihovo kreacijo.

Primer kreiranja novega računa:

```
POST /receipt
```

PUT

Metoda PUT je pomembna za shranjevanje v podatkovno bazo. Z njo lahko kreiramo tako nove entitet kot posodabljamo že obstoječe. Tudi metoda PUT mora biti idempotentna, kar je tudi glavna razlika v primerjavi z metodo POST, ki tega ne zahteva. V primeru, da entiteta že obstaja, bo vedno prišlo do njene zamenjave.

Primer modifikacije obstoječega računa z enoličnim identifikatorjem 1:

```
PUT /recipt/1
```

DELETE

Zahtevek za odstranitev vira, ki je lahko takojšen ali asinhron.

Primer izbrisa računa z enoličnim identifikatorjem 1:

```
DELETE /recipt/1
```

3.6 Git

Git je trenutno najbolj uporabljen in razširjen sistem za verzioniranje kode. Uporabnikom omogoča hranjenje kode na oddaljenem strežniku, pregled zgodovine, v primeru napačnih implementacij pa omogoča povrnitev sistema na eno izmed predhodnih verzij. Razil ga je Linus Torvalds leta 2005. Glavno prednost pred konkurenti (SVN) zagotavlja uporaba distribuiranega sistema, kar uporabniku dovoljuje dostop in uporabo, ne glede na točko nahajanja.

3.7 Maven

Apache Maven je orodje, ki se uporablja za gradnjo in opisovanje projekta [8]. Temelji na XML-datoteki POM (angl. Project Object Model), ki definira vse strukture in vtičnike za uspešno delovanje programa. Razvili so ga leta 2004 (Apache Software Foundation) in je eno izmed najbolj priljubljenih orodij razvijalcev programske opreme. Definira tako zgradbo programske opreme kot vse potrebne knjižnice za razvoj.

3.7.1 Konvencija pred konfiguracijo

Izraz konvencija pred konfiguracijo (angl. convention over configuration) je paradigma, ki skuša zmanjšati razvojnikove odločitve pri arhitekturnih lastnostih projekta, s čimer omogoča več časa za sam razvoj aplikacije. Vse našteto Maven uporabniku omogoča s preprosto izgradnjo POM-datoteke.

3.7.2 POM

Datoteka XML vsebuje vse potrebne metainformacije o projektu in procesna navodila za izgradnjo (ime projekta, verzijo, odvisne knjižnice in vtičnike).

Nekaj izmed njih je podrobneje opisanih v nadaljevanju.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>si.albin</groupId>
    <artifactId>Receiptiver</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>Receiptiver</name>
    <description>Receiving receipts </description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.3.0.RELEASE</version>
        <relativePath />
    </parent>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>si.albin.common</groupId>
            <artifactId>apiObjects</artifactId>
            <version>0.0.1-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-mongodb</artifactId>
        </dependency>
```

Slika 3.1: Datoteka pom.xml

3.7.3 Življenjski cikel aplikacije Maven

Maven temelji na konceptu gradnje projekta s pomočjo življenjskega kroga. To pomeni, da je proces gradnje vnaprej pripravljen in definiran. Glavni trije življenjski krogi, ki jih vsebuje, so: osnovni (angl. default – namestitev na strežnik), počisti (angl. clean – skrbi za čiščenje zgrajene kode) in stran (angl. site – kreiranje dokumentacije). Vsak življenjski krog gradi ena izmed naslednjih faz:

- validiraj (angl. validate) – preveri pravilnost izgrajenega projekta,
- zgradi (angl. compile) – sestavi osnovno kodo projekta,

- testiraj (angl. test) – testira zgrajeno kodo s pomočjo enega izmed ogrodij Unit,
- pakiraj (angl. package) – zapakira kodo v želeni format, npr. .jar,
- integracijski test (angl. integration-test) – uporaben pri testiranju storitvenih sistemov,
- preveri (angl. verify) – preveri pravilnost zgrajenega paketa,
- namesti (angl. install) – namesti zgrajeno kodo in potrebne module v lokalni repozitorij,
- naloži (angl. deploy) – namesti zgrajeno kodo in potrebne module v oddaljen repozitorij.

3.8 HTML

HTML (angl. HyperText Markup Language) je standard, razvit leta 1993, ki definira strukturo spletnih strani. Jezik je spisan s pomočjo HTML-elementov (značk), ki so obdani s kotnimi oklepaji (⟨body⟩ vsebina ⟨/body⟩). Z njimi lahko na spletno stran dodajmo slike, video in/ali besedilo. Značke iz nabora knjižnice SVG (angl. Scalable Vector Graphics) pa uporabniku omogočajo gradnjo kompleksnejših vektorskih struktur. Spletni brskalniki so prilagojeni tako, da HTML-značke interpretirajo in jih uporabniku predstavijo na razumljiv način.

3.8.1 HTML 5.0

HTML 5.0 je trenutno najnovejša verzija, ki je z definicijo novih značk močno izboljšala možnosti programerjev. Programerjem omogoča dodajanje kontrolnih atributov, semantičnih elementov in/ali multimedije. Poleg značk SVG pa dodaja značko canvas, ki še dodatno izboljšuje kakovost predstavitve grafičnih vsebin.

3.9 Javascript

Javascript je skriptni programski jezik, ki je bil razvit za potrebe DOM-manipulaciije HTML-elementov. Najpogosteje se izvede šele na strani uporabnika (spletni brskalnik). Podpira velik nabor funkcionalnosti strukturiranega programiranja (if, while, switch stavki itd.), glavna razlika pa se pojavlja pri definiranju spremenljivk. Java na primer uporabniku omogoča definiranje spremenljivk glede na njihove tipe (int, boolean, double itd.), pri čemer pa so pri Javascriptu vsi tipi spremenljivk definirani kot var.

3.9.1 JQuery

JQuery je popularna knjižnica, ki jo je razvil John Resig leta 2006. Razvita je bila z namenom združevanja kode Javascript v programerju prijaznejšo in bolj funkcionalno obliko. Glavne prednosti uporabe knjižnice so: jasnost in jedrnatost zapisane kode, odpravljanje nezdružljivosti spletnih brskalnikov ter razširljivost.

Programerju pa poleg uporabnosti nudi tudi nov nabor funkcionalnosti:

- hitrejša in preprostejša manipulacija strukture DOM,
- izboljšana podpora lovljenja dogodkov (angl. events),
- efekti in animacije,
- AJAX,
- JSON-razčlenjevanje,
- vtičniki (angl. plugins).

3.10 Velocity

Velocity je javanska tehnologija, ki se uporablja za procesiranje predlog v HTML-strukturo. Sama predloga je opisano v datoteki s končnico .vm. Razvijalcu ponuja velik nabor predefiniranih struktur in združevanje podobnih

delov kode v strukturo, imenovano macro, ki je kasneje dostopna v vseh .vm datotekah.

```
#(macro vrednost_seznama $vrednost)
   <li>$vrednost</li>
#end
<ul>
#foreach($vrednost in $vrednosti)
   #vrednost_seznama($vrednost)
#end
</ul>
```

Primer za podane vrednosti (1, 2, 3) bi generiral naslednjo HTML-kodo:

```
<ul>
<li>1</li>
<li>2</li>
<li>3</li>
</ul>
```

3.11 CSS

CSS (angl. Cascading Style Sheets) skrbi za definicijo podobe spletne strani. Najpogosteje se nahaja v zunanji datoteki (končnica .css), ki je kasneje vključena v HTML-dokument. Jezik je razvilo podjetje W3C, podpira pa ga večina spletnih brskalnikov.

Poglavje 4

Načrtovanje in implementacija aplikacije

V nadaljevanju bo predstavljen proces izdelave vseh funkcionalnosti, potrebnih za nemoteno delovanje sistema. Lotili smo se agilnega procesa izgradnje našega sistema. To pomeni, da smo najprej definirali zgodbe, ki jih moramo dokončati. Glavne zgodbe, ki smo jih definirali, so bile naslednje:

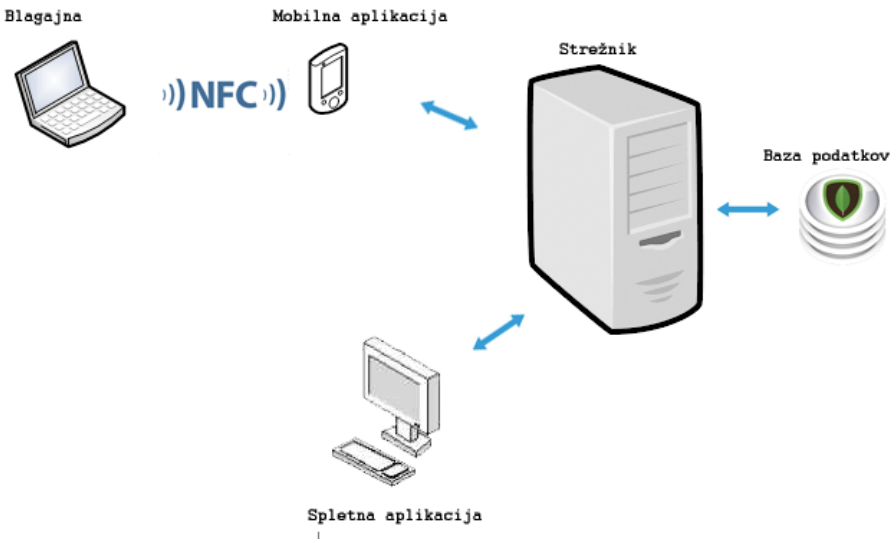
- podatkovni del (definicija potrebnih kolekcij MongoDB in ustrezne POJO-preslikave),
- strežniški del (postavitev aplikacije Spring Boot z ustreznimi API-konfiguracijami in poslovno logiko),
- implementacija mobilne aplikacije (podoba, strežniški klici in NFC-komunikacija),
- simulacija blagajne in NFC-komunikator,
- spletni del aplikacija (podoba, strežniški klici in manipulacija DOM-strukture s pomočjo Javascripta).

Po končani definiciji smo posamezno zgodbo razdelili na manjše podzgodbe in jim glede na zahtevnost dodelili okvirno število točk (angl. story

points). Kot nam narekuje metodologija dela SCRUM, smo podzgodbam dodelili določeno časovno obdobje (angl. sprint), znotraj katerega smo jih zaključili. V našem primeru smo zaradi drugih obveznosti (študij, delo) dolžino sprinta definirali kot 2-tedenski termin s tremi urami dela na dan.

4.1 Arhitektura celotnega sistema

Za lažjo predstavo delovanja sistema in potrebne infrastrukture smo si najprej skicirali način povezovanja med posameznimi gradniki sistema.



Slika 4.1: Arhitektura celotne storitve

Kot prikazuje slika 4.1, je naš sistem zgrajen iz petih glavnih gradnikov.

Centralni del sistema je strežnik, ki komunicira s spletno aplikacijo in mobilno aplikacijo ter pridobiva oziroma shranjuje podatke v podatkovni del našega sistema (v bazo podatkov).

Naslednji večji del sistema je mobilna aplikacija, ki mora zadostiti trem glavnim nalogam. Prva naloga je pridobivanje računov iz NFC-terminala, za kar je potrebna ustrezna infrastrukturo, ki lovi morebitne NFC-signale in omogoča njihov prenos. Druga naloga je shranjevanje informacij v sistemsko bazo podatkov (SQLite), saj se lahko zgodi, da ima uporabnik med prenosom

računa izklopljen podatkovni prenos podatkov ali pa ga uporablja samo v primeru brezžične povezave. Tretja naloga mobilne aplikacije pa je skrb za ustrezno sinhronizacijo med lokalnimi podatki in strežnikom (klici REST).

Sledi spletni del sistema, ki mora uporabniku zagotavljati varnost dostopa (avtentikacija) in pregled računov z ustreznimi analizami. Pri tem moramo biti pozorni še na ustrezno manipulacijo podatkov, ki jo izvede brskalnik, saj bomo le tako uporabniku nudili optimalno uporabniško izkušnjo.

Zadnji del sistema pa je implementacija vmesnika, ki zagotavlja uspešno posredovanje računa, izdanega na blagajni, s pomočjo NFC-terminala do mobilne naprave.

4.2 Strežniški del aplikacije

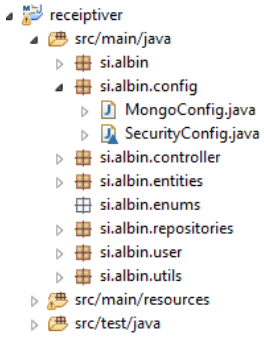
Strežniški del smo postavili s pomočjo sistema Spring Boot. V začetnem (inicialnem) oknu smo definirali želene knjižnice, s pomočjo katerih smo dobili potrebno strukturo datoteke .pom za uspešno izgradnjo aplikacije. Uporabili smo naslednje začetne pakete Spring:

- Web Web (uporaba principa MVC skupaj z vgrajenim strežnikom Tomcat, ki omogoča pakiranje aplikacije v datoteko .jar),
- MongoDB (knjižnice za povezovanje z bazo in ustrezne anotacije za dostop),
- Velocity (ustrezen prevajalnik predlog .vm v HTML-strukturo),
- Security (knjižnice za zagotavljanje varne avtentikacije v sistem),
- Jersey JAX-RS (infrastruktura za uspešno komunikacijo REST).

4.2.1 Direktorijska struktura in paketi aplikacije

Po uspešnem uvozu pripravljenega projekta v Eclipse smo definirali vse potrebne pakete, ki smo jih potrebovali za boljšo preglednost in strukturo apli-

kacije. Dodatno smo ustvarili poseben modul Maven, v katerem smo hranili skupne objekte, ki se bodo uporabljali za prenos podatkov.



Slika 4.2: Direktorijska struktura projekta

4.2.2 Podatkovni del – MongoDB

Podatkovni del našega sistema je zaradi uporabe baze MongoDB in njenega principa kolekcij zelo preprost. Za našo osnovno verzijo aplikacije sta tako potrebni samo dve glavni kolekciji in nekaj manjših za določene izračune.

Kolekcija Account nam definira uporabnika sistema z vsemi pripadajočimi lastnostmi, potrebnimi za avtentikacijo.

```
{
    "_id" : ObjectId("56a36a6add586c88ca953c53"),
    "username" : "janko123",
    "firstname" : "Janez",
    "lastname" : "Kranjski",
    "password" : "e4a695bf7686c6caf469f16c47",
    "email" : "janez.kranjski@gmail.com",
    "phoneNumber" : "+38670123456",
    "role" : "ROLE_USER"
}
```

Slika 4.3: JSON-predstavitev kolekcije Account v bazi

Kolekcija Receipt pa je potrebna za mapiranje podatkov z računa, pri čemer je zelo pomembno, da računu dodamo ustrezen enolični identifikator

(uporabniško ime), saj bo potreben za nadaljnjo pridobivanje in urejanje računa.

Slika 4.4: JSON predstavitev kolekcije Receipt v bazi

Konfiguracija dostopa in mapiranje podatkov

Spring Boot omogoča zelo enostavno konfiguracijo dostopa do podatkovne baze z modifikacijo datoteke *application.properties*. V datoteki smo definirali lokacijo baze ter uporabniško ime in geslo za dostop. Nato smo ustvarili še javanski razred, ki prebere podane vrednosti in ustvari dejanska zrna (angl. bean), s katerimi pridobivamo oziroma zapisujemo podatke v bazo.

Konfiguracija javanskega razreda:

```
@Configuration
@EnableMongoRepositories(basePackages = "si.albin.repositories")
public class MongoConfig {
```

```
    @Value("${mongo.database}")
    public String mongoDatabase;

    @Value("${mongo.schema}")
    public String mongoSchema;

    @Bean
       public MongoDbFactory mongoDbFactory() throws
           UnknownHostException {
          return new SimpleMongoDbFactory(new
              MongoClient(mongoDatabase), mongoSchema);
        }

       @Bean
       public MongoTemplate mongoTemplate() throws
           UnknownHostException {
             MongoTemplate template = new
                 MongoTemplate(mongoDbFactory(), mongoConverter());
             return template;
       }
}
```

4.2.3 Avtentikacijski del – Spring Security

Varnost naše aplikacije smo zagotovili z uporabo knjižnic, ki jih omogoča paket Spring Security. Za pravilno delovanje smo morali konfigurirati:

- javanski razred, ki predstavlja strukturo objekta v bazi podatkov (strukturo JSON, prikazano na sliki 4.3),
- storitveni razred, ki je potreben za pridobivanje in zapis uporabnikov v podatkovno bazo,
- kontroler (angl. controller) za prestrezanje zahtevkov REST in pravilno

preusmeritev,

- konfiguracijo Spring, potrebno za ustrezno mapiranje zgornjih razredov.

```
@Configuration
@EnableWebSecurity
class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Bean
    public UserService userService() {
        return new UserService();
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth)
        throws Exception {
           auth
            .eraseCredentials(true)
            .userDetailsService(userService())
            .passwordEncoder(passwordEncoder());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
               .antMatchers("/css/**", "/less/**").permitAll()
                .antMatchers("/", "/resources/**",
                    "/signup").permitAll()
                .anyRequest().authenticated()
                .and()
            .formLogin()
                .loginPage("/signin")
```

```
                .permitAll()
                .failureUrl("/signin?error=1")
                .loginProcessingUrl("/authenticate")
                .defaultSuccessUrl("/", true)
                .and()
            .logout()
                .logoutUrl("/logout")
                .permitAll()
                .logoutSuccessUrl("/signin?logout")
                .and()
            .rememberMe()
                .rememberMeServices(rememberMeServices())
                .key("remember-me-key")
                .and()
            .csrf().disable();
    }
}
```

Zgoraj predstavljeni javanski razred *SecurityConfig* razširja *WebSecurityConfigurerAdapter* in je anotiran z *@EnableWebMvcSecurity*, kar omogoča integracijo z modelom Spring Security in možnost razširitve, glede na potrebe razvijalca.

Z metodama *configure* smo definirali uporabo lastne baze podatkov in različna dovoljenja oziroma preusmeritve pri dostopih do aplikacije.

4.3 Mobilna Android aplikacija – NFC Receipt

Mobilno aplikacijo smo razvili za sistem Android, za sam razvoj pa smo uporabljali razvojno orodje Android Studio. Razvoj mobilne aplikacije je bil razdeljen na več delov.

Ker je bila to prva mobilna aplikacija, ki smo jo izdelali, smo se naj-

prej s pomočjo različnih vodičev spoznali s samim konceptom in definirali direktorijsko strukturo projekta.

Sledila je izdelava podobe, ki smo jo definirali s pomočjo orodja Adobe Illustrator. Samo podobo smo nato z uporabo XML-datotek (layout) in statičnih podatkov prenesli v našo aplikacijo.

Seznaniti se je bilo treba tudi s samo knjižnico za NFC-komunikacijo, s pomočjo katere smo zgradili infrastrukturo, potrebno za prenašanje računov med blagajno in našo napravo Android. Pridobljeni račun smo najprej shranili v lokalno bazo SQLite, v primeru vključenega prenosa podatkov pa smo z REST-klicem strežnika račun zapisali v podatkovno bazo.

4.3.1 Arhitektura podobe mobilne aplikacije

Ob prvem zagonu aplikacije se uporabniku prikaže okno za avtentikacijo (uporabniško ime in geslo), ki podatka v primeru uspešno izvedene avtentikacije shrani v sam telefon. Pri naslednjih zagonih aplikacije tako uporabnik dostopa neposredno do menija, seveda pa ima vedno na voljo možnost odjave. Zaradi enostavnosti in preglednosti smo uporabniku omogočili le naslednje funkcije: možnost pregleda vseh računov, podrobnosti računa, osnovno analizo porabe po mesecih in možnost sinhronizacije s spletno aplikacijo, ki pa ponuja širši nabor funkcionalnosti.

Pregled vseh računov

V sekciji pregleda računov lahko uporabnik pregleduje tako lokalno shranjene račune kot že sinhronizirane račune, shranjene v podatkovni bazi. Funkcija iskanja mu/ji omogoča hiter pregled računov glede na podano ime trgovine, datum, ceno nakupa in lokacijo shranjenega računa. S klikom na želeni račun se uporabniku prikaže novo okno z vsemi podrobnostmi nakupa.

Slika 4.5: Vsi računi

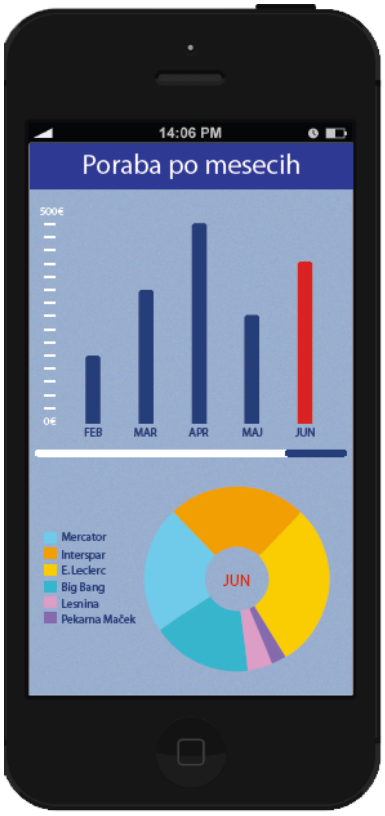
Pregled podrobnosti računa

Podrobnosti računa uporabniku nudijo hiter pregled informacij nakupa. Drsno okno omogoča sprehajanje med nakupljenimi izdelki, gumb ,,sinhroniziraj" pa ob vklopljenem prenosu podatkov račun posreduje strežniku, ki ga obdela in shrani v podatkovno bazo.

Slika 4.6: Podrobnosti računa

Mesečna analiza nakupov

Pomembnejša funkcija aplikacije je analiza nakupov. Tu ima uporabnik na voljo dva diagrama za pregled porabe. V stolpčnem diagramu so prikazani skupni mesečni stroški, tortni diagram pa prikazuje razdelitev stroškov glede na trgovino nakupa.



Slika 4.7: Analiza nakupov

Implementacija XML-datoteke

Sistem Android uporablja za definicijo podobe XML-datoteke (navadno imenovane layout). Elemente lahko definiramo s pomočjo izbire v meniju ali pa v datoteko layout dodamo njihovo XML-definicijo. Za potrebe naše aplikacije smo definirali *bodyLayout.xml* kot ogrodje (glava, noga), na katerega smo s pomočjo značke *include* dodajali želeno vsebino.

Eden izmed težavnejših elementov je bil drsni sistem (GridView). Ta za ustrezno delovanje potrebuje:

- XML-predstavitev elementa,
- zunanjo definicijo (layout), ki predstavlja strukturo elementov v samem drsniku,
- javanski razred za ustrezno mapiranje pridobljenih podatkov.

```
private static class ReceiptListAdapter extends
    ArrayAdapter<Receipt> {
       private List<Receipt> receipts;
       public ReceiptListAdapter(Context context, int resource,
           int textViewResourceId,
                                 List<Receipt> receipts) {
           super(context, resource, textViewResourceId, receipts);
           this.receipts = receipts;

       }
       @Override
       public View getView(int position, View convertView,
           ViewGroup parent) {
           View v = super.getView(position, convertView, parent);
           TextView t1 = (TextView)
               v.findViewById(R.id.receiptShop);
           t1.setText(receipts.get(position).getShopId());
           TextView t2 = (TextView)
               v.findViewById(R.id.receiptDate);
           t2.setText(receipts.get(position).getCreationTime().toString());
           TextView t3 = (TextView)
               v.findViewById(R.id.receiptAmount);
           t3.setText(receipts.get(position).getSumAmount());
           return v; }
   }
```

4.3.2 Komunikacija NFC

Za uspešno uporabo NFC-protokola na naši mobilni napravi smo morali omogočiti ustrezna dovoljenja. S prvim dovoljenjem `android.permission.NFC` smo dovolili nadzor nad strojno opremo NFC, dovoljenje `android.permission.READ_PHONE_STATE` pa nam je omogočilo prepoznavanje sprememb na naši mobilni napravi.

S pomočjo knjižnice NFCAdapter smo v primeru NFC-aktivnosti prebrali podatke v bajtih in jih pretvorili v niz. Niz (XML) smo nato s pomočjo knjižnice JAXB pretvorili v vnaprej pripravljen objekt, ki je predstavljal naš račun. V primeru uspešnega prenosa smo od uporabnika zahtevali pritisk gumba, s katerim je NFC-adapterju vrnil uspešen rezultat. V primeru napake pri prenosu podatkov pa smo vrnili rezultat z neuspešnim statusom in od naprave zahtevali vnovično pošiljanje podatkov.

Glavne metode, ki opisujejo zgoraj opisane korake:

```
public void checkstatus(String action, Intent intentNFC) {
       if (NfcAdapter.ACTION_NDEF_DISCOVERED.equals(action)) {
           String receiptXML = retrieveRecieptStr(intentNFC);

           if(StringUtils.isNotBlank(receiptXML)) {
               Receipt recievedReceipt =
                   convertToObject(receiptXML, Receipt.class);

               if(recievedReceipt != null) {
                   HomeActivity.this.returnMessageWithStatus(Status.SUCCESS);
                   startNewIntent(recievedReceipt);
               }
           }

     HomeActivity.this.returnMessageWithStatus(Status.ERROR);
     restartIntent();
       }
   }
```

```
 public String retrieveRecieptStr(Intent intentNFC) {
Tag tag = (Tag)
    intentNFC.getParcelableExtra(NfcAdapter.EXTRA_TAG);
      byte[] id =
          intentNFC.getByteArrayExtra(NfcAdapter.EXTRA_ID);
      Parcelable[] rawMsgs =
          intentNFC.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
      ArrayList<NdefMessage> messages = new
          ArrayList<NdefMessage>();

if (rawMsgs != null) {
        NdefMessage[] msgs = new NdefMessage[rawMsgs.length];
          for (int i = 0; i < rawMsgs.length; i++) {
                messages.add((NdefMessage) rawMsgs[i]);
                NdefRecord record =
                    messages.get(i).getRecords()[i];
                String message =
                    getTextData(record.getPayload());
                return message;
          }
      }

      return null;
 }

public void returnMessageWithStatus(String status) {
      NdefRecord recordToSend = createRecord(status);
      NdefMessage messageToSend = createMessage(recordToSend);
      mAdapter.enableForegroundNdefPush(this, messageToSend);
 }
```

4.3.3 Pošiljanje podatkov na strežnik

Za komunikacijo s strežnikom smo uporabili Springovo knjižnico *RestTemplate*, ki nam omogoča pošiljanje HTTP-zahtevkov. Pri pošiljanju zahtevkov smo poleg osnovnega URL-naslova v zahtevek dodali še uporabniško ime in uporabniški žeton (angl. token), ki nam je služil kot enolični identifikator. Podatka smo pridobili iz lokalno shranjenega podatkovnega vnosa, kreiranega pri prvi uspešni prijavi.

Za izvajanje klicev strežnika smo ustvarli nov javanski razred, ki je razširjal generični razred *AsyncTask⟨Params, Progress, Result⟩*. Slednji je pomemben predvsem za boljšo uporabniško izkušnjo, saj se za izvedbo klica ustvari nova nit, ki uporabniku, ne glede na to, ali je klic uspel ali ne, omogoči nemoteno uporabo aplikacije.

Primer dela kode, ki se uporablja za pridobivanje vseh uporabniških računov:

```
private class HttpGetReceiptList extends AsyncTask<Void, Void,
    List<Receipt>> {

        @Override
        protected List<Receipt> doInBackground(Void... params) {
            try {
                final String url = SERVER_URL + username;
                RestTemplate restTemplate = new RestTemplate();
                restTemplate.getMessageConverters().add(new
                    MappingJackson2HttpMessageConverter());
                Receipt[] receipts = restTemplate.getForObject(url,
                    Receipt[].class);
                return Arrays.asList(receipts);
            } catch (Exception e) {
                Log.e("Error while accessing server: ",
                    e.getMessage(), e);
            }
```

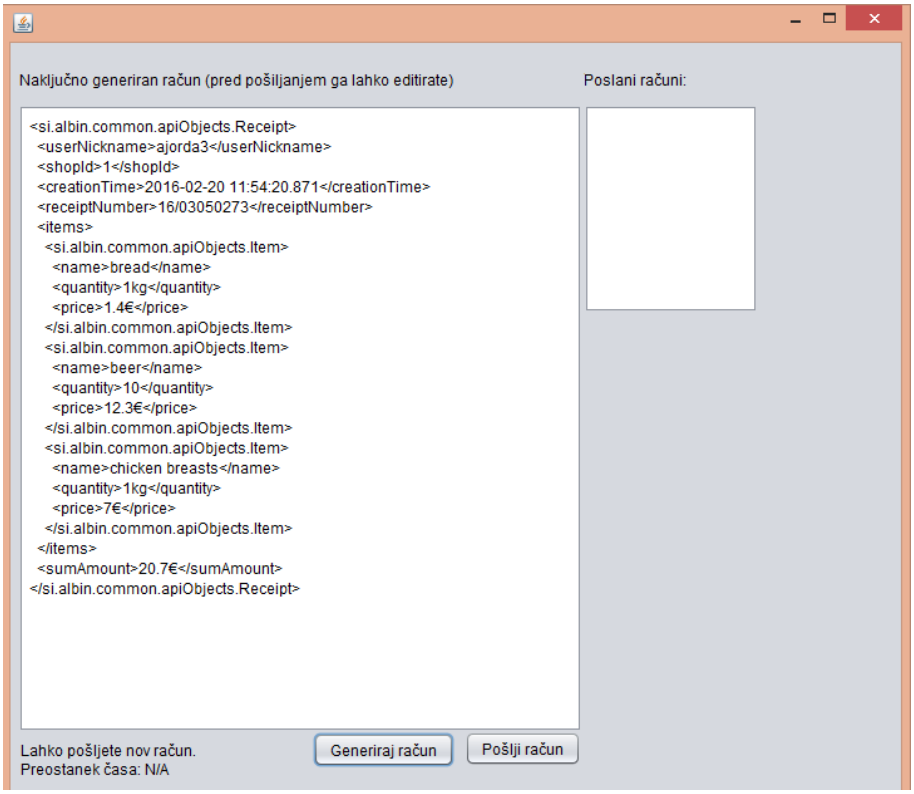
```
        return null;
    }
```

4.4 Simulacija blagajne in NFC-komunikator

Za potrebe razvoja in testiranja NFC-komunikacije smo zgradili preprosto namizno aplikacijo. Aplikacija je namenjena generiranju naključnih računov in njihovemu pošiljanju s pomočjo protokola NFC. Za izgradnjo simulacije smo uporabili javansko tehnologijo Swing, generiranje računov pa smo izpeljali z uporabo knjižnice XStream. Napolnili smo razred Receipt z naključnimi podatki in ga serializirali v niz podatkov (XML). V naslednjem podpoglavju sta predstavljeni grafična podoba aplikacije in implementacija vmesnika za pošiljanje računov.

4.4.1 Grafična podoba simulacije

Podobo namizne aplikacije smo najprej definirali v programu Phosotoshop. Podoba aplikacije je precej minimalistična, kar poskrbi za boljšo uporabniško izkušnjo in hitrejši razvoj. Glavni funkciji simulacije sta generiranje računa in možnost urejanja. Vse to nam je omogočeno s pritiskom na gumb ,,generiraj račun". Na uporabniškem vmesniku se nam prikaže naključno generirana vsebina, ki je predstavljena v XML-strukturi. V prikazanem oknu lahko nato po želji prirejamo generirane podatke in dodajamo novo artikle, kot prikazuje slika 4.8.



Slika 4.8: Namizna aplikacija za pošiljanje računov.

4.4.2 Vmesnik za NFC-komunikacijo

Pošiljanje računa mobilni napravi smo izvedli s pomočjo odprtokodne knjižnice nfctools, ki omogoča uporabo komunikacije vsak z vsakim (P2P). S pritiskom na gumb ,,pošlji računše ustvari nova nit, ki v zanki pošilja sporočilo. V primeru uspešno oddanega sporočila se zanka zaključi, adapter pa prične s čakanjem na odgovor. Za potrebe ustrezne komunikacije smo razširili logiko čakanja s števcem, ki po eni minuti zaključi delo in izpiše sporočilo o neuspehu. V primeru, da je mobilna naprava vrnila status OK, pa se sporočilo shrani na listo uspešno poslanih računov.

Del opisane komunikacije prikazuje koda spodaj:

```
public NfcManager(boolean initiatorMode, NfcPushData view) {
        nfcView = view;
        this.initiatorMode = initiatorMode;
        ndefPushLlcpService = new NdefPushLlcpService(new
            NdefListener() {
            @Override
            public void onNdefMessages(Collection<Record> messages) {
                ArrayList<Record> listRecord = (ArrayList<Record>)
                    messages;
                TextRecord textRecord = (TextRecord)
                    listRecord.get(0);
                nfcView.displayResultOfAnswer(textRecord.getText());

            }
        });
        llcpOverNfcip = new LlcpOverNfcip(new
            LlcpConnectionManagerFactory() {
         @Override
         protected void
             configureConnectionManager(LlcpConnectionManager
             connectionManager) {
            connectionManager
            .registerWellKnownServiceAccessPoint(LlcpConstants.COM_ANDROID_NPP,
                 ndefPushLlcpService);
         }
      });
    }
    public void addMessages(String message) {
        ndefPushLlcpService.addMessages(convertToList(new
            TextMimeRecord("text/ndefTransaction", message)), new
            NdefPushFinishListener() {
            @Override
            public void onNdefPushFinish() {
```

```
                nfcView.sendingStatusToView(true);
            }

            @Override
            public void onNdefPushFailed() {
                nfcView.sendingStatusToView(false);
            }
        });
    }
    public void displayResultOfAnswer(String answer) {
       if (answer.equals("Success")) {
          JOptionPane.showMessageDialog(this, "Receipt was
              successfully accepted");
          historyMessage.add(tempMessage);
          DefaultListModel<String> listModel = new
              DefaultListModel<String>();
          for (String message : historyMessage) {
             listModel.addElement(message);
          }
          listMessageAlreadySend.setModel(listModel);
          restartNFC();
       } else {
          JOptionPane.showMessageDialog(this, "Sending receipt
              failed.");
          restartNFC();
       }
    }
```

4.5 Implementacija spletnega dela

Z izdelavo spletne strani smo povzeli funkcije, implementirane za mobilno aplikacijo, in dodali nekatere zahtevnejše, ki pa bi bile pri mobilni aplikaciji

prezahtevne za uporabnika. Na strežniškem delu smo definirali kontrolerje (angl. controller), ki skrbijo za prestrezanje HTTP-zahtevkov in njihovo poslovno obdelavo ter vračajo HTML-strukturo z vsebovanimi stili in skriptnimi dodatki. Za izgradnjo HTML-strukture smo uporabili predlogo Velocity, ki nam je z združevanjem kode občutno skrajšala čas razvoja. Podobo spletne aplikacije smo definirali v programskem orodju Photoshop in jo v aplikacijo prenesli s pomočjo kaskadne stilske predloge LESS, ki smo jo s pomočjo vtičnika Maven prevedli v CSS, ki ga kasneje interpretira spletni brskalniki in prikaže uporabniku.

4.5.1 Osnovna stran

Osnovna stran je namenjena hitremu pregledu vseh uporabniških funkcij. V zgornjem delu je prikazanih pet uporabniku najpomembnejših podatkov:

- število izdanih računov v tekočem mesecu,
- število izdelkov, ki jim v tekočem mesecu poteče garancija,
- število različnih trgovin, v katerih je uporabnik v tem mesecu nakupoval,
- izdelke, ki jih je uporabnik definiral kot nujne nakupe in še niso bili zabeleženi,
- preostanek mesečnega proračuna, ki ga je definiral uporabnik.

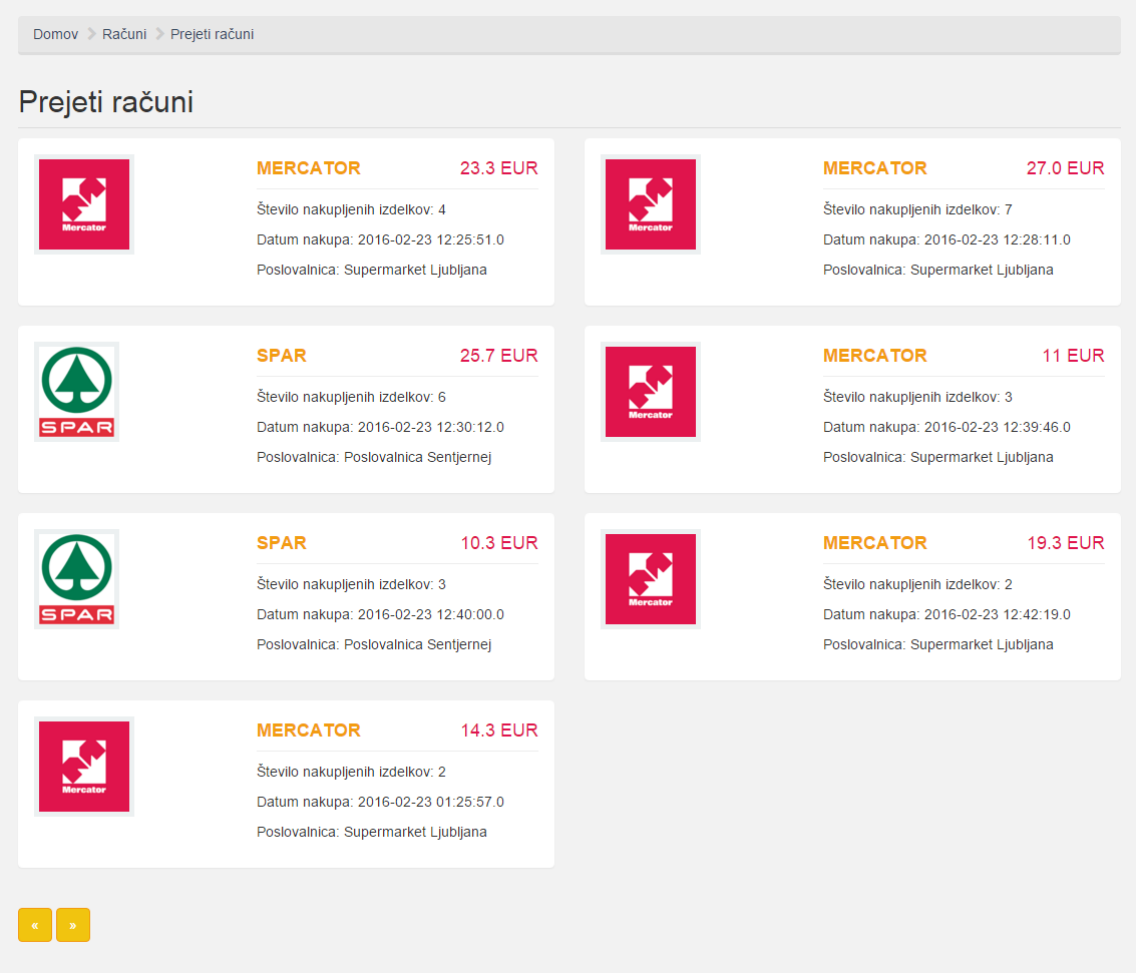
V spodnjem delu osnovne strani uporabniku prikazujemo diagram, ki povzema njegove/njene mesečne porabe. Z drsnikom lahko podatke približa na dnevno raven porabe oziroma oddalji na letno. Zaradi večje preglednosti prikazujemo skupno porabo v vseh trgovinah v izbranem obdobju.

4.5.2 Lista računov

Lista računov uporabniku omogoča pregled vseh računov, ki jih je prejel s pomočjo naše mobilne aplikacije. Omogočeno mu je iskanje po vseh elemen-

tih računa. Samo iskanje sprožimo z uporabo asinhronih klicev (AJAX), ki pridobijo podatke iz podatkovnega dela in zamenjajo samo želeno vsebino brez ponovnega nalaganja strani. Uporabnik ima tudi možnost filtriranje glede na vrsto izdelkov in poslovalnico, kjer je opravil storitev.

Zaradi preglednosti in hitrega iskanja prikazujemo le glavne podatke računa. Ob kliku na zbrani račun pa se uporabniku v novem oknu prikažejo tudi druge podrobnosti.



Slika 4.9: Lista računov

Poglavje 5

Sklepne ugotovitve

V diplomskem delu smo razvili sistem za prejemanje elektronskih računov. Postavili smo mobilno aplikacijo, spletno aplikacijo in simulacijo blagajne z grafičnim vmesnikom za lažje generiranje in pošiljanje računov. Mobilna aplikacija skrbi za prejemanje računov s pomočjo protokola NFC in prejete podatke sinhronizira s spletno aplikacijo. NFC-ogrodje za komunikacijo je na platformah Android zelo dobro podprto in dokumentirano, kar nam je omogočilo hiter in učinkovit razvoj. Veliko več težav pa smo imeli z izgradnjo simulacije blagajne in s pošiljanjem računov. Edina uporabna javanska knjižnica, ki smo jo našli na spletu, je bila odprtokodna knjižnica NFC tools, ki je sicer vsebovala nekaj (močno zastarelih) primerov, žal pa ni vsebovala ustrezne dokumentacije za lažje razumevanje. Za uspešen prenos podatkov je bilo treba knjižnico močno dodelati in urediti. Zahtevnejšim uporabnikom smo ugodili z izgradnjo spletne aplikacije, ki je namenjena kompleksnejšim analizam porabe. Svoje nakupe lahko uporabnik pregleduje v grafičnem diagramu, dodatno pa lahko določi maksimalno mesečno porabo glede na trgovino, aplikacija pa ga ob prekoračitvi porabe ustrezno opozori.

5.1 Možne nadgradnje obstoječega sistema

Pri razvoju sistema smo se srečali z mnogimi idejami za razširitev in nadaljnji razvoj, ki pa jih zaradi pomanjkanja časa nismo uspeli realizirati.

Ena izmed njih je razširitev mobilne aplikacije z možnostjo prejemanja garancijskih listov, kar bi močno olajšalo njihovo shranjevanje. Uporabnika bi aplikacija opozorila o bližajočem se poteku garancije in mu prikazala bližnje poslovalnice, ki omogočajo popravila.

Prav tako bi lahko poseben razdelek spletne aplikacije namenili izdelkom na računu. Ceno enakih izdelkov bi lahko primerjali glede na različne ponudnike in časovna obdobja. Uporabnik bi lahko dodatno definiral nakupovalni seznam, aplikacija pa bi ga napotila v najcenejšo poslovalnico.

Možnosti nadgrajevanj je še mnogo, vendar bi bilo treba najprej pridobiti zaupanje ene od poslovalnic in spraviti sistem na trg, torej do uporabnikov.

5.2 Spremna misel

Razvijanje sistema je bilo zabavno in poučno. Seznanili smo se z novimi tehnologijami in močno izboljšali razporejanje časa s pomočjo agilne metodologije SCRUM. Menimo, da bi lahko v bodoče sistem predstavili eni izmed večjih trgovin in s tem uporabnikom zagotovili novo orodje za bistveno lažje urejanje osebnih financ, obenem pa naravi prihranili marsikatero drevo.

Literatura

[1] R. Want, "Near field communication," *IEEE Pervasive Computing*, no. 3, pp. 4–7, 2011.

[2] G. Madlmayr, J. Langer, C. Kantner, and J. Scharinger, "Nfc devices: Security and privacy," in *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, pp. 642–647, IEEE, 2008.

[3] E. Burnette, "Hello, android: introducing google's mobile development platform," Pragmatic Bookshelf, 2009.

[4] Wikipedia, "Mongodb – wikipedia, the free encyclopedia." `https://en.wikipedia.org/w/index.php?title=MongoDB&oldid=706251576/`, 2015. [Dostopano 25. 11. 2015].

[5] K. Chodorow, "Mongodb: the definitive guide," O'Reilly Media, Inc., 2013.

[6] C. Walls, "Developing applications with java and uml," Addison-Wesley Longman Publishing Co., Inc., 2001.

[7] S. Francia, "Rest vs soap, the difference between soap and rest." `http://spf13.com/post/soap-vs-rest/`, 2016. [Dostopano 20. 2. 2016].

[8] J. F. Smart *et al.*, "An introduction to maven 2," *JavaWorld Magazine. http://www.javaworld.com/javaworld/jw-12-2005/jw-1205-maven.html*, 2005. [Dostopano 14. 1. 2016].

[9] Wikipedia, "Near field communication – wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Near_field_communication&oldid=707170347/, 2016. [Dostopano 15. 1. 2016].

[10] Wikipedia, "Html – wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=HTML&oldid=707579665/, 2016. [Dostopano 20. 2. 2016].

[11] S. F. Conservancy, "Documentation." https://git-scm.com/doc/, 2015. [Dostopano 8. 10. 2015].

[12] S. Chapman, "What is javascript?." http://javascript.about.com/od/reference/p/javascript.htm/, 2015. [Dostopano 8. 10. 2015].

[13] H. W. Lie and B. Bos, "Cascading style sheets, level," *W3C, www.w3.org/pub/www/TR/REC-CSS1-961217*, pp. 1–86, 1996. [Dostopano 5. 2. 2016].

[14] C. Haase and R. Guy, "Filthy rich clients: Developing animated and graphical effects for desktop java applications," Addison-Wesley Professional, 2007.

[15] C. Walls, "Spring in action," Manning Publications, 2011.